EF150396349US

METHOD AND APPARATUS TO BOOT A NON-UNIFORM-MEMORY-ACCESS (NUMA) MACHINE

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates to multiprocessor computer architectures and more specifically to Non-Uniform-Memory-Access (NUMA) machines.

2. Description of Related Art:

A Symmetric Multiprocessing (SMP) architecture contains multiple Central Processing Units (CPUs) that reside in one cabinet and share the same memory. This architecture allows SMP systems to provide scalability, depending on the user's needs, such as transaction volume.

SMP systems can contain from two to 32 or more CPUs. However, if one CPU within a SMP system fails, the entire system fails. To guard against CPU failure, redundancy can be provided using two or more SMP systems in a cluster. In this way, if one SMP system in the cluster fails, the others can continue to operate and compensate for the lost system.

A single CPU usually boots the SMP system and loads the operating system, which brings the other CPUs online. Because the CPUs in a SMP system share the same memory, there is only one operating system and one instance of the application in memory. SMP systems are particularly advantageous whenever processes can be overlapped. For example, multiple applications may be run simultaneously.

Another example is multithreading, which comprises concurrent operations within a single application.

Non-Uniform Memory Access (NUMA) architecture is a multiprocessing architecture in which memory is separated into close and distant banks. Like SMP, a NUMA machine comprises multiple CPUs sharing a single memory. However, in NUMA, local memory located on the same processing board as the CPUs is accessed faster than shared memory located on other processing boards.

A SMP system memory map architecture can be structured to support a NUMA machine with individual SMP systems connected with the SMA NUMA adapters. When each SMP system is a standalone machine, the system node ID field in the PIR register of each Giga-Processor (GP) is always set to 0. The firmware always uses the memory map corresponding to the system ID0 to perform Remote Input/Output (RIO) configuration.

When several SMP systems are connected to form a NUMA machine, the individual SMP systems will have to be set up properly with respect to the NUMA memory map for them to function correctly in the NUMA machine. While maintaining the firmware function to boot the SMP system in standalone mode, it is desirable that the same system firmware image can provide a method to configure the entire NUMA machine in order to boot all individual SMP machines for normal operation in NUMA mode.

In addition, it is also desirable that this new method will manage to boot the NUMA machine without incurring significant boot time spent on performing RIO configurations, i.e. concurrently performing RIO configurations of each system node.

SUMMARY OF THE INVENTION

The present invention provides a method, apparatus and program for booting a non-uniform-memory-access (NUMA) machine. The invention comprises configuring a plurality of standalone, symmetrical multiprocessing (SMP) systems to operate within a NUMA system. A master processor is selected within each SMP; the other processors in the SMP are designated as NUMA slave processors. A NUMA master processor is then chosen from the SMP master processors; the other SMP master processors are designated as NUMA slave processors. A unique NUMA ID is assigned to each SMP that will be part of the NUMA system. The SMPs are then booted in NUMA mode in one-pass with memory coherency established right at the beginning of the execution of the system firmware.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 depicts a pictorial representation of a data processing system in which the present invention may be implemented;

Figure 2 depicts a block diagram of a data processing system is shown in which the present invention may be implemented;

Figure 3 depicts a flowchart illustrating the process of SMP system configuration in accordance with the present invention;

Figure 4 depicts a flowchart illustrating the process of configuring the host processor and memory in accordance with the present invention; and

Figure 5 depicts a flowchart illustrating the process of booting a NUMA system in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference now to the figures and in particular with reference to **Figure 1**, a pictorial representation of a data processing system in which the present invention may be implemented is depicted in accordance with a preferred embodiment of the present invention. A computer **100** is depicted which includes a system unit **110**, a video display terminal **102**, a keyboard **104**, storage devices **108**, which may include floppy drives and other types of permanent and removable storage media, and mouse **106**. Additional input devices may be included with personal computer **100**, such as, for example, a joystick, touchpad, touch screen, trackball, microphone, and the like. Computer **100** also preferably includes a graphical user interface that may be implemented by means of systems software residing in computer readable media in operation within computer **100**.

With reference now to **Figure 2**, a block diagram of a data processing system is shown in which the present invention may be implemented. Data processing system **200** is an example of a computer, such as computer **100** in **Figure 1**, in which code or instructions implementing the processes of the present invention may be located. Data processing system **200** employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus architectures such as Accelerated Graphics Port (AGP) and Industry Standard Architecture (ISA) may be used. Processors **202**, **204**, **206** and main memory **210** are connected

to PCI local bus **208** through PCI bridge **212**. PCI bridge **212** also may include an integrated memory controller and cache memory for processors **202**, **204**, and **206**. Additional connections to PCI local bus **208** may be made through direct component interconnection or through add-in boards. In the depicted example, local area network (LAN) adapter **214**, small computer system interface SCSI host bus adapter **216**, and expansion bus interface **218** are connected to PCI local bus **208** by direct component connection. In contrast, audio adapter **220**, graphics adapter **222**, and audio/video adapter **224** are connected to PCI local bus **208** by add-in boards inserted into expansion slots. Expansion bus interface **218** provides a connection for a keyboard and mouse adapter **224**, modem **226**, and additional memory **228**. SCSI host bus adapter **216** provides a connection for hard disk drive **230**, tape drive **232**, and CD-ROM drive **234**. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors.

A single operating system runs on processors **202, 204**, and **206** and is used to coordinate and provide control of various components within data processing system **200** in **Figure 2**. The operating system may be a commercially available operating system such as Windows 2000, which is available from Microsoft Corporation. An object oriented programming system such as Java may run in conjunction with the operating system and provides calls to the operating system from Java programs or applications executing on data processing system **200**. "Java" is a trademark of Sun Microsystems, Inc. Instructions for the operating system, the object-oriented programming system,

and applications or programs are located on storage devices, such as hard disk drive **230**, and may be loaded into main memory **210** for execution by processors **202, 204** and **206.**

Those of ordinary skill in the art will appreciate that the hardware in **Figure 2** may vary depending on the implementation. Other internal hardware or peripheral devices, such as flash ROM (or equivalent nonvolatile memory) or optical disk drives and the like, may be used in addition to or in place of the hardware depicted in **Figure 2**. Also, the processes of the present invention may be applied to a multiprocessor data processing system.

The depicted example in **Figure 2** and above-described examples are not meant to imply architectural limitations.

An Symmetric Multiprocessing (SMP) system memory map is structured to support a Non-Uniform-Memory-Access (NUMA) configuration by connecting individual SMP machines. When individual machines are used in stand alone mode, the system ID in the processor is set to zero. The present invention uses the same firmware to configure the system in SMP and NUMA mode. The HSC for the NUMA system will instruct each SMP system's service processor to set up the system in NUMA mode based on the NUMA configuration setup variables stored in each SMP system's Non-Uniform Random Access Memory (NURAM).

The NUMA machine has one or more Hardware System Consoles (HSC) to manage and configure the NUMA machine. Each SMP system has its own built-in Common Service

Processor (CSP). The HSC connects to all CSPs in some communication links.

Referring to **Figure 3**, a flowchart illustrating the process of SMP system configuration is depicted in accordance with the present invention. In this stage of the startup process, the HSC communicates with the SMPs and assigns each SMP an ID which designates the SMP's place within the NUMA machine. Each CSP of the system maintains two Non-Volitile Random Access Memory (NVRAM) byte-variables for NUMA configuration setup:

- NUMA _mode_flag: If NUMA_mode_flag = 0, the system is in standalone SMP mode. Otherwise, the mode, and the value of the variable, is a bitmask to indicate the presence of the system nodes within the NUMA machine.
- NUMA_node_id: This contains the system node ID for the SMP system where this NVRAM variable exists.

The HSC instructs all CSPs to set up the NUMA_mode_flag with the proper mask value for the NUMA machine (step **301**), and then assigns and instructs each CSP to set up its unique NUMA_node_id (step **302**). After the NUMA_mode_flag and NUMA_node_id are set, the HSC sends a power-on command to each node's CSP to power on its attached SMP system (step **303**).

Referring to **Figure 4**, a flowchart illustrating the process of configuring the host processor and memory is depicted in accordance with the present invention. Each node's CSP configures and tests the host processors (step **401**), and then configures and tests the host memory (step **402**). After Power On Self Test (POST), the memory will be

assigned a system memory base address, but can be non-contiguous. For example, the memory of a SMP system can be non-contiguous with numeral memory regions, but cannot go beyond its 256G size.

After the host processors and memory are configured and tested, the CSP configures and tests the NUMA memory (step **403**) and sets the node ID to X (step **404**). The CSP then sets up the base addresses for all Remote Input/Output (RIO) hubs by using a system memory map reserved for node ID X (step **405**).

Then CSP temporarily maps local system memory starting at the address zero (step **406**), and then loads the system firmware image into its local system memory (**407**). From there, the CSP permanently maps the local system memory at the starting address Y that the memory map has reserved for node ID X (step **408**). Once the local system memory has been mapped, the CSP informs the HSC of the version of loaded firmware image (step **409**) and then waits for the HSC to confirm that the loaded system firmware image is the same as the other node's firmware image (step **410**).

The CSP configures its NUMA adapters to connect the system into the final NUMA machine (step **411**), and initializes the local nodal-time-base register (step **412**).

Making use of the logical partition hardware of the Giga-processor, each node's CSP starts all GP processors with:

- MSR[SF]= 1, which tells the processor to start executing code in 64-bit mode.
- MSR[HV]= 0, which places the processor in a

logical partition environment.

- HID0[19]= 1, which indicates that the timebase function of the processor is operating in NUMA mode. It also serves as a NUMA firmware flag, so that the firmware must follow the NUMA execution path.
- HID4[AS/RS]= 1, which selects the processor running in the RS/6000 server machine's environment.
- HID4[RMOR]. Set this register (node's memory base address) based in the equation: System memory base address (Y) = NUMA_node_id * 256G.
- HID4[RMLR]. Set this register (real mode limit register) to encode 1G real mode address-size enabling.
- HID4[LPID]= 0, which sets the logical partition ID to 0, the default value.
- PIR[23,25]= NUMA_node_id. Set this register to the node_id of the SMP system where the processor resides.
- NIA= 0x100. This sets the system firmware to execute at its entry point.

Step (**413**).

The CSP sets all host processors' program counters to the system firmware's entry point (step **414**), and releases all host processors so that they can start executing the system firmware (step **415**). The above setups allow the system firmware images at individual nodes to run as if it is loaded at memory address 0.

Referring now to **Figure 5**, a flowchart illustrating the process of booting a NUMA system is depicted in accordance with the present invention. The process begins when each node selects a master processor which will configure the node's RIO (step **501**). Each system node's firmware has implemented a master software semaphore. All processors from the same system node will compete to obtain this semaphore. The winner successfully obtaining the semaphore becomes the nodal master processor. The processors that do not become the master nodal processor are referred to as slave processors. The nodal slave processors wait for the nodal master processor to initiate one-on-one handshakes (step **503**), at which time the slave processors set their timebase (TB) registers (step **504**). The slave processors then switch to a hyper-visor environment to become NUMA slave processors (step **505**). Hyper-visor environment is an executing environment wherein the processors have unrestricted access to the system's hardware. All NUMA slave processors then execute firmware residing in node 0, and wait for the NUMA master processor to allow them to proceed further (step **506**).

The nodal processor that is selected to be the nodal master processor must configure the RIO hubs of the local node (step **507**). After finishing RIO configuration, the nodal master processor synchronizes its TB register with the nodal TB (step **508**) and then handshakes with each nodal slave processor so that each slave processor will also synchronize its TB register (step **509**). After the handshake, the nodal slave processors leave the partition

environment and go to hyper-visor environment, as described in step **505**. The slave processors continue executing the same code which will now be fetched from the memory of the system node ID0. These nodal slave processors will now be the NUMA slave processors of the NUMA machine, and wait for the NUMA master processor to initiate one-on-one handshakes. Each nodal master processor then sets up the node's Global Queue Interrupt Mask (GQIRM) registers (step **510**).

Each nodal master processor leaves the partition environment and goes to global hyper-visor environment and hyper-visor memory area to compete to be the NUMA master processor (step **511**). The system firmware also implements the NUMA master software semaphore. Since all nodal master processors all switch to the hyper-visor environment, the NUMA master software semaphore of the firmware for the system node ID0 is the competition target. Again, the winner successfully obtaining this NUMA master semaphore becomes the NUMA master processor. If a nodal master processor fails to be the NUMA master processor, it will wait for one-on-one handshaking with the NUMA master processor and then become a NUMA slave processor (step **513**).

The processor selected as the NUMA master processor gathers all updated nodal RIO structures, all nodal memory status, and all nodal processor status (step **514**). The NUMA master processor then executes open firmware and creates the open firmware device tree for the entire NUMA machine (step **515**). The NUMA master processor goes on to boot the NUMA machine as if it is logically a giant SMP machine. This comprises loading the operating system

(OS) into NUMA system memory and transferring control to the OS (step **516**). The NUMA master processor runs the OS code to take all of the NUMA slave processors to their destination in the OS (step **517**).

The method of the present invention establishes the memory coherency among SMP system nodes before the execution of the System firmware. This allows the NUMA machine to be started with coherent system memory, thus avoiding the non-coherence of the traditional two-pass method of the prior art.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media, such as a floppy disk, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs, and transmission-type media, such as digital and analog communications links, wired or wireless communications links using transmission forms, such as, for example, radio frequency and light wave transmissions. The computer readable media may take the form of coded formats that are decoded for actual use in a particular data processing system.

The description of the present invention has been presented for purposes of illustration and description,

and is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.